PERSISTENCE: I/O DEVICES

Shivaram Venkataraman
CS 537, Spring 2020
CHECK: 1,2,3

1. Can you hear me?

2. Can you see the slides and the annotations?

3. Can you write a question on the chat?
Project 4a: Out today! Due April 2nd, 10pm

Grades: Project 3 grades out!
Midterm grades end of the week (hopefully)
AGENDA / LEARNING OUTCOMES

How does the OS interact with I/O devices?

What are the components of a hard disk drive?
RECAP
OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization
2. Concurrency
3. Persistence

Make each application believe it has each resource to itself CPU and Memory

Provide mutual exclusion, ordering
What good is a computer without any I/O devices?
   keyboard, display, disks

We want:
   - **H/W** that will let us plug in different devices
   - **OS** that can interact with different combinations
HARDWARE SUPPORT FOR I/O

CPU

Memory

Memory Bus (proprietary)

General I/O Bus (e.g., PCI)

Graphics

Peripheral I/O Bus (e.g., SCSI, SATA, USB)
CANONICAL DEVICE

Device Registers

- Status
- COMMAND
- DATA

OS reads/writes to these
while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
  wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
  wait for interrupt;
INTERRUPTS VS. POLLING

Are interrupts always better than polling?

Fast device: Better to spin than take interrupt overhead
  – Device time unknown? Hybrid approach (spin then use interrupts)

Flood of interrupts arrive
  – Can lead to livelock (always handling interrupts)
  – Better to ignore interrupts while make some progress handling them

Other improvement
  – Interrupt coalescing (batch together several interrupts)
# Protocol Variants

Status checks: polling vs. interrupts

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other special-purpose chips</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DATA TRANSFER COSTS

CPU
1 1 1 1 1 1 c c c 2 2 2 2 2 2 1 1

Disk
1 1 1 1 1 1
PROGRAMMED I/O VS. DIRECT MEMORY ACCESS

PIO (Programmed I/O):
- CPU directly tells device what the data is

DMA (Direct Memory Access):
- CPU leaves data in memory
- Device reads data directly from memory
while (STATUS == BUSY)  
;
Write data to DATA register  
Write command to COMMAND register  
while (STATUS == BUSY)  
;
# Protocol Variants

Status checks: polling vs. interrupts

<table>
<thead>
<tr>
<th>Status</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td>Extra RAM</td>
<td>Other special-purpose chips</td>
</tr>
</tbody>
</table>

PIO vs DMA
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
SPECIAL INSTRUCTIONS VS. MEM-MAPPED I/O

Special instructions
- each device has a port
- in/out instructions (x86) communicate with device

Memory-Mapped I/O
- H/W maps registers into address space
- loads/stores sent to device

Doesn’t matter much (both are used)
## Protocol Variants

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other special-purpose chips</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status checks: polling vs. interrupts

PIO vs DMA

Special instructions vs. Memory mapped I/O
DEVICE DRIVERS

- Application
  - POSIX API [open, read, write, close, etc.]
    - File System
      - Generic Block Interface [block read/write]
        - Generic Block Layer
          - Specific Block Interface [protocol-specific read/write]
            - Device Driver [SCSI, ATA, etc.]
VARIETY IS A CHALLENGE

Problem:
  – many, many devices
  – each has its own protocol

How can we avoid writing a slightly different OS for each H/W combination?

Write device driver for each device

Drivers are **70%** of Linux source code
If you have a fast non-volatile memory based storage device, which approach would work better?

What part of a device protocol is improved by using DMA?
HARD DISKS
HARD DISK INTERFACE

Disk has a sector-addressable address space
   Appears as an array of sectors

Sectors are typically 512 bytes

Main operations: reads + writes to sectors

Mechanical and slow (?)
Motor connected to spindle spins platters

Rate of rotation: RPM

10000 RPM → single rotation is 6 ms
Surface is divided into rings: **tracks**

Stack of tracks (across platters): **cylinder**
Tracks are divided into numbered sectors
Heads on a moving arm can read from each surface.
READING DATA FROM DISK

Rotational delay
READING DATA FROM DISK

Seek Time

Rotates this way

Remaining rotation

Spindle

Seek

9

21

33

8

20

32

7

19

18

17

6

5

4

3

2

1

0

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32
TIME TO READ/WRITE

Three components:
Time = seek + rotation + transfer time
Seek cost: Function of cylinder distance
   Not purely linear cost
   Must accelerate, coast, decelerate, settle
   Settling alone can take 0.5 - 2 ms

Entire seeks often takes 4 - 10 ms
Average seek = 1/3 of max seek

Depends on rotations per minute (RPM)
   7200 RPM is common, 15000 RPM is high end

Average rotation?

Pretty fast: depends on RPM and sector density.
100+ MB/s is typical for maximum transfer rate
What is the time for 4KB random read?

<table>
<thead>
<tr>
<th></th>
<th>Cheetah 15K.5</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Average Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>16/32 MB</td>
</tr>
<tr>
<td>Connects via</td>
<td>SCSI</td>
<td>SATA</td>
</tr>
</tbody>
</table>
NEXT STEPS

Advanced disk features
Scheduling disk requests

Project 4a: Out tonight
Grades: Project 2b, 3, midterm by tomorrow!